Amplicon sequencing (16S and ITS) processing platform

Thomas Gurry - thomasgurry@gmail.com

1 Introduction

This document describes the process of going from raw 16S or ITS data to processed data (OTU tables, oligotypes, etc.). This is orchestrated by the script `raw2otu.py`. The platform is designed in such a manner that the user interfaces with a single script, `Master.py`, which takes as input the path to the folder containing the dataset. Each dataset folder must contain a machine-readable text file, called a summary file, which gives instructions to `Master.py`. The format of these summary files, and all files required for 16S or ITS processing, are described in the section below.

2 Preparing your dataset for processing

2.1 Raw FASTQ files

FASTQ files have different ASCII encodings for the quality characters. It is often ASCII base 33 or ASCII base 64. Check this using `usearch -fastq_chars yourFASTQfile.fastq` and specify the encoding in the summary file. If left unspecified, it will default to base 33.

The pipeline is designed to take a single FASTQ file in which all reads have been concatenated. In the event that the reads have already been demultiplexed and each individual FASTQ file corresponds to a given sample, instead of specifying the FASTQ file (cf. 'Case 3' below). The pipeline will split them for parallelization and recombine them for OTU calling.

2.2 Metadata

Metadata is uploaded as is to S3 for storage with the raw data. Ideally, we want to capture as much of the metadata as possible in the database and into file objects describing the data (e.g. OTU tables in BIOM format). We can produce a file with three columns: `sample_ID`, `disease_label` and `keywords`. `sample_ID` corresponds to the name ID of the sample as is listed in the OTU table. `disease_label` is a specific label used for each subject and can be more general than disease. Note that none of these metadata are currently used by the pipeline, so you can process your data without knowing anything other than the

sample barcode. This format is merely recommended to facilitate standardized storage and of course necessary for downstream analysis.

2.3 Summary files

This file, named `summary_file.txt`, is a machine-readable, **tab-delimited** file that must accompany any dataset directory when uploaded to the cloud. It should be found in the highest directory level for the dataset directory in the S3 bucket. It is a text file with descriptors for the data and paths to all relevant datafiles within the directory. It can include a True or False flag for whether any associated raw 16S/ITS data has already been processed.

The order in which items are listed between the lines `#16S_start` and `#16S_end` (for 16S) and between lines `#ITS_start` and `#ITS_end` (for ITS) does not matter.

Note that any white space in the summary file examples below corresponds to a single tab character.

2.3.1 List of 16S and ITS attributes

See the table below.

2.3.2 Case 1: raw FASTQ file of 16S sequences, still includes primers and barcodes

The simplest case is if you have the following files: a raw FASTQ file; a file specifying the map between barcode sequences and IDs; and a file specifying the primers used. Your summary file would look something like this:

```
DATASET_ID myDataset

#16S_start
RAW_FASTQ_FILE myData.fastq
ASCII_ENCODING ASCII_BASE_33
PRIMERS_FILE primers.txt
BARCODES_MAP barcodes_map.txt
BARCODES_MODE 2
METADATA_FILE metadata.txt
PROCESSED False
#16S_end
```

Note that you must also specify the place where barcodes are to be found, i.e. either in the ">" sequence ID lines (mode 1) or in the sequences themselves (mode 2). The `PROCESSED` flag tells the processing instance that the dataset needs to be processed into OTU tables.

Attribute	Description
RAW_FASTQ_FILE	Raw FASTQ file name/path within the dataset directory
RAW_FASTA_FILE	Raw FASTA file name/path if raw data is in FASTA format
RAW_FASTQ_FILES	For demultiplexed datasets where samples are separated into separate FASTQ files. Filename of two column file containing FASTQ filenames in first column and sample IDs in the second column.
ASCII_ENCODING	ASCII quality encoding in FASTQ. Supports either 'ASCII_BASE_33' or 'ASCII_BASE_64'. Set to 33 if unspecified.
PRIMERS_FILE	Filename/path to primers file.
BARCODES_MAP	Filename/path to barcodes map file.
BARCODES_MODE	'1' = barcodes in sequence ID, '2' = barcodes in sequences themselves.
BARCODES_SEPARATOR	Separator character. See description in Case 2 below.
METADATA_FILE	Filename/path to metadata file.
PROCESSED	True/False flag for whether data have already been processed. Set to 'False' for processing to proceed.
TRIM_LENGTH	Length to which all sequences should be trimmed. Defaults to 101 if unspecified.
QUALITY_TRIM	Minimum quality score allowed. Any bases below this will be trimmed. Defaults to 25 if unspecified.
MIN_COUNT	Minimum sequence count in dereplication across all samples. Defaults to 10 if unspecified (i.e. sequences with fewer than 10 occurrences in the entire dataset will not be considered downstream).
OTU_SIMILARITY	Integer specifying the percent similarity desired in OTU clustering. Defaults to 97 if unspecified.
RDP_CUTOFF	Desired probability cut-off for Ribosomal Database Project assignments. Assignments at each taxonomic level will be evaluated and those with a lower probability than this cutoff will be labeled as unidentified.
GG_ALIGN	Specific to 16S sequences. True/False flag for whether GreenGenes alignments are desired. Set to 'True' if unspecified.
UNITE_ALIGN	Specific to ITS sequences. True/False flag for whether UNITE alignments are desired. Set to 'True' if unspecified.

2.3.3 Case 2: raw FASTQ file of ITS sequences, primers and barcodes have been removed

In the case where the 'raw' data has actually had primers and barcodes previously removed, the sample IDs must be listed in the sequence ID lines of the FASTQ file. When the pipeline

removes barcodes itself and replaces them with sample IDs, individual sequence reads for a given `sampleID` will be annotated as `sampleID;1, sampleID;2`, etc., where we note here that the `BARCODES_SEPARATOR` is ';'. However, in a dataset where the barcodes have previously been removed, you will have to look into the FASTQ file to check the 'separator' character. Your summary file would look something like this:

```
DATASET_ID myDataset

#ITS_start
RAW_FASTQ_FILE myData.fastq
ASCII_ENCODING ASCII_BASE_33
PRIMERS_FILE None
BARCODES_MAP None
BARCODES_SEPARATOR ;
METADATA_FILE metadata.txt
PROCESSED False
#ITS_end
```

2.3.4 Case 3: multiple demultiplexed raw FASTQ or FASTA files of 16S sequences, each file corresponding to a single sample

Sometimes sequencing data are available in a demultiplexed form, where the reads for each sample are split into separate files. Many datasets in the SRA, for example, are available in this form. In this case, you can create a **two-column, tab-delimited file** where the first column lists the filename and the second column lists the corresponding sample ID. Note that paths should be relative paths *within the current directory*, e.g. `datafiles/file1.txt` for a folder called `datafiles` within the current directory. In the summary file, the `RAW_FASTQ_FILE` line becomes `RAW_FASTQ_FILES` (plural), and instead refers to this filename. If your files are FASTA rather than FASTQ, simply use `RAW_FASTA_FILES` (also plural). For a filename `fastq_filemap.txt`, your summary file would look something like this:

```
DATASET_ID myDataset

#16S_start
RAW_FASTQ_FILES fastq_filemap.txt
ASCII_ENCODING ASCII_BASE_33
PRIMERS_FILE primers.txt
BARCODES_MAP barcodes_map.txt
BARCODES_MODE 2
METADATA_FILE metadata.txt
PROCESSED False
```

```
#16S_end
```

3 Running the pipeline

Once you have placed all relevant files and folders, including the summary file into a single folder, you can call the script `Master.py` taking the path to this folder as input. `Master.py` which will parse the summary file and launch the appropriate scripts to process your request. Suppose the path to the folder containing the dataset is `/home/ubuntu/dataset_folder`, and contains a summary file where you specified the dataset ID as 'myDataset', you would run the following command from anywhere:

```
nohup python ~/scripts/Master.py -i /home/ubuntu/dataset_folder &
```

Processing happens in a folder within the `proc` folder, with path `/home/ubuntu/proc/myDataset_proc_16S` or `/home/ubuntu/proc/myDataset_proc_ITS`, depending on the amplicon being analyzed. Results are put in `/home/ubuntu/processing_results/`.

4 Pipeline description

4.1 Quality control

Within the results folder, there is a subfolder called `quality_control`, which contains various plots diagnostic of dataset quality. Currently, the pipeline outputs:

- Histogram showing distribution of read lengths, taken from the first 100,000 reads in the raw FASTQ file.
- Bar chart showing number of reads per sample.
- File showing percentage of reads thrown out at each processing step (`processing_summary.txt`).

4.2 OTU calling

The pipeline produces various OTU tables. All tables are constructed from the set of raw dereplicated reads. These are a set of reads in FASTA format that appear in the file `datasetID.raw_dereplicated.fasta` and correspond to the set of unique sequences present in the raw data.

4.2.1 *de novo* OTU tables

The dereplicated reads are clustered to within the specified sequence similarity percentage cut-off (`OTU_SIMILARITY`, default: 97) using `usearch`. Individual reads are assigned either as the OTU centroid or as a match. Centroids are labeled as 'OTU_ID.0' and matches count

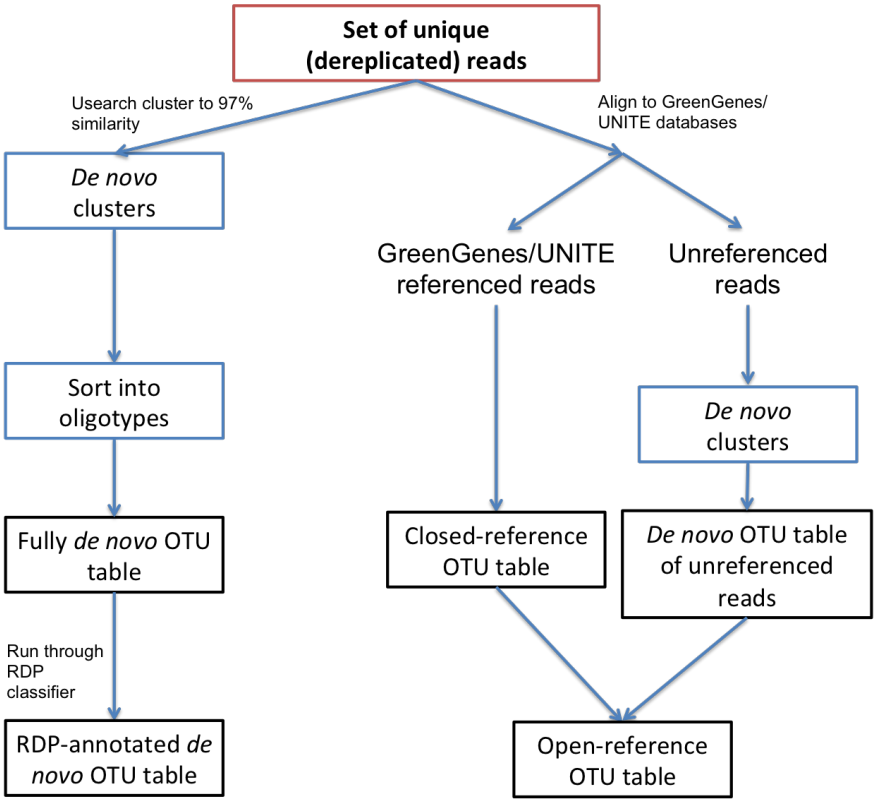

Figure 1: Schematic of OTU calling pipeline.

from 'OTU_ID.1' onwards. These are separate oligotypes within the OTU 'OTU_ID'. All oligotype counts are then collapsed to their respective OTU, resulting in a fully *de novo* OTU table which can be found in the filename `datasetID.otu_table.*.denovo` where '*' gets replaced by the OTU similarity cut-off.

4.2.2 Closed-reference OTU table

There are two types of database-referencing that are outputed from the pipeline by default. The first is assignments from the Ribosomal Database Project (RDP), which returns probabilistic assignments. The default probability cut-off below which an assignment is labeled as unidentified at a given taxonomic level is 0.8, but this can be set using a summary file parameter `RDP_CUTOFF`. This OTU table can be found in the results sub-folder called 'RDP'.

The dereplicated reads are also aligned to a standard database (GreenGenes in the case of 16S sequences and UNITE in the case of ITS sequences). In the case of GreenGenes, the database is determined based on the specified OTU similarity cut-off: e.g. `97_otus.fasta` for `OTU_SIMILARITY` set to 97). The alignment is performed using `usearch`, and considers the top 10 hits. Consensus assignments are then produced for the top 1, top 3, top 5 and top 10 hits (where a taxonomic level is only assigned a latin name if the top N hits from GreenGenes agree), and the corresponding OTU tables are output. Thus, the OTU table called `datasetID.otu_table.*.gg.consensus10` contains latin names which are formed from a minimum consensus of the top 10 hits for each taxonomic level, where '*' gets replaced by the OTU similarity cut-off. Levels are left unidentified (e.g. 's__') if the consensus requirement is not met.

Note that the database referencing process is one of the slower steps in the pipeline, so if you only care about RDP, you can skip the GreenGenes assignments steps by setting the parameter `GG_ALIGN` to `False` in the summary file. Similarly, for ITS sequences, you can skip the UNITE assignments steps by setting the parameter `UNITE_ALIGN` to `False`.

4.2.3 Open-reference OTU table

If any reads in the dereplicated sequences do not align to GreenGenes/UNITE to within the specified similarity cut-off, they are clustered with the desired similarity cut-off using `usearch`, and appended to the GreenGenes/UNITE closed-reference table to produce a full, open-reference OTU table (which combines GreenGenes-referenced and *de novo* OTUs) at the desired similarity cut-off, with filename `datasetID.otu_table.gg.*.open_ref`.

4.2.4 Oligotypes

The OTU calling process automatically assigns separate IDs to unique sequences within OTUs (termed 'oligotypes' by some). These offer a greater level of granularity. For ex-

ample, if sequences A, B and C are all the same OTU, called OTU1 (and the centroid is sequence A), these would be relabeled 'OTU1.0', 'OTU1.1' and 'OTU1.2'.

There are currently two such tables output: the full *de novo* oligotype table (where sequences are labeled according to a fully *de novo* OTU clustering process) - `datasetID.oligotype_table.class` - and the reduced oligotype table which only describes those sequences which do not map to GreenGenes -

`datasetID.denovo_oligotype_table.classic`.

4.3 Taxon abundances and diversities

...

4.4 PiCRUST

OTU tables are sent off to a second node for metagenome prediction using PiCRUST. It will return predicted abundances of the different KEGG modules, including abundances of pathways collapsed at each different level of abstraction in the KEGG hierarchy. These abundances are encoded in BIOM format tables and can be found in the results directory. Stacked bar plots of the different KEGG module abundances on a per sample basis, as produced by QIIME, are also added to the results directory.

4.5 Downstream analysis

...

5 Resulting 16S features

The resulting 16S features are returned in a features object, computed using the `Features.py` module. This object can be found in pickled form in the results directory, under the filename `pickled_features.pkl`. The pickled object can be loaded in Python as follows:

```
import pickle

with open('pickled_features.pkl', 'r') as fid:
    features = pickle.load(fid)
```

The resulting `features` object has the full feature vectors for each sample stored in `features.feature_vectors`, which is a `dict` whose keys are the sample IDs (the same as those specified in the provided metadata file, and in the resulting OTU table). Thus, the feature vector for sample 'A' can be found in `features.feature_vectors['A']`. The name of the feature at index i of the feature vectors can be accessed from `features.feature_descriptions[i]`. The full list of features includes:

- OTU abundances.
- Oligotype abundances (in the form OTU1.1, OTU 1.2, OTU 1.3, ..., where OTU1.1 represents oligotype 1 of OTU 1).
- KEGG module abundances, at levels 1, 2, 3 and 4.
- L/R phylogenetic features.

6 Source code - Python modules

6.1 Modules

- `Formatting.py` - Module to house miscellaneous formatting methods, e.g. conversion from classic dense format to BIOM format, OTU table transposition, etc.
- `QualityControl.py` - Methods for quality control diagnostics on a dataset.
- `preprocessing_16S.py` - Methods and wrappers for raw 16S sequence data processing.
- `CommLink.py` - Communications module between the various processing nodes. Links processing node, PiCRUST server and QIIME/plotting server. Manages inbox listeners for each type.
- `Taxonomy.py` - Methods for taxonomy-related feature extraction and analytics. Includes functions for things like: -adding latin names to a GreenGenes-referenced OTU table -collapsing abundances at different taxonomic levels
- `Phylogeny.py` - Methods for phylogenetic feature extraction, e.g. left/right (LR) abundance ratios at each node of a phylogenetic tree.
- `Analytics.py` - Generic statistical analysis tools, e.g. Wilcoxon tests across all available taxa.
- `Regressions.py` - Performs different types of regressions *en masse*.
- `UserInterface.py` - Routines for preparing JSON files and other required files for use on the UI/dashboard.
- `DepositDB.py` - Methods for depositing data into the database.

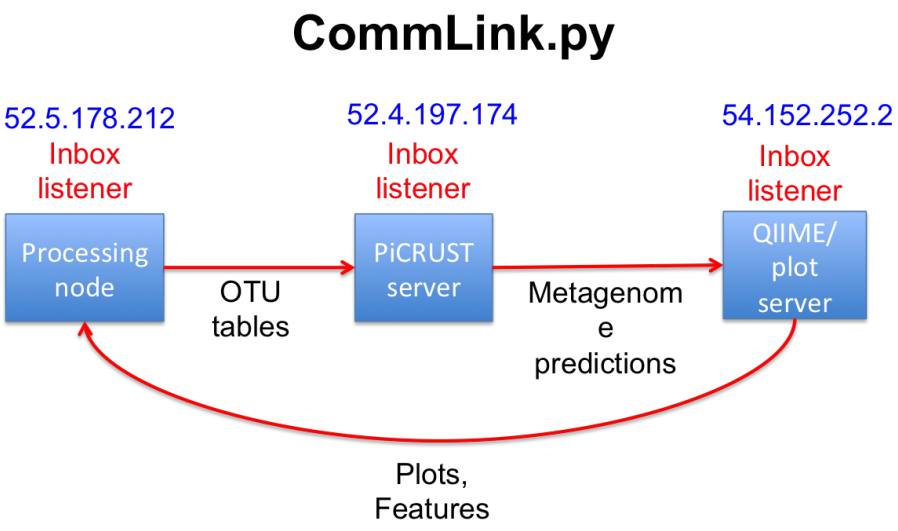

Figure 2: Work flow managed by CommLink.py.

6.2 Scripts and routines

- `Master.py` - Master script that calls relevant processing pipelines, e.g. `raw2otu.py`.
- `raw2otu.py` - Pipeline for converting raw 16S FASTQ sequence files to OTU tables. Handles parallelization requirements in these processing steps automatically. Takes as input a directory that contains a summary file and the raw data.

7 Troubleshooting

Standard error (`stderr`) and standard out (`stdout`) are directed to the directory `/home/ubuntu/logs`. Errors will appear in the `stderr` file for the corresponding datasets. In addition, file outputs from each step of the pipeline can be found in `/home/ubuntu/proc/datasetID_proc_16S`, which can help to diagnose errors. Common sources of errors or difficulties include:

- Bad formatting of the summary, barcodes, or primers file. There should be no white spaces, and columns should be tab-delimited. If you created the file in Excel, it may have carriage return or other non-linux formatting characters that introduce difficulties. Safest is to go from a previous summary file template, or to create it manually.
- Typos in a 16S attribute key-value pair in the summary file.

- Incorrect ASCII encoding specified. Check whether 33 or 64 using `usearch8 -fastq_chars yourFASTQfile.fastq`. Default is 33, so if left unspecified and the file is ASCII base 64, quality trimming will fail.
- For ITS sequences, the RDP classifier often runs out of RAM when loading the UNITE database. This is usually what happened if you find `RDP_classification.txt` to be empty in the relevant folder in `/home/ubuntu/proc`. If you rerun the pipeline several times, have checked everything else and keep encountering this problem, you may need a node with more RAM to do your processing.
- If you specify a file containing a list of FASTQ/FASTA files for the raw reads, make sure the file paths are relative to the directory hosting the summary file.